Igor Golyanov

Automation Tool for Tasks in Computer Graphics Production

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

14 March 2013

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Igor Golyanov Automation Tool for Tasks in Computer Graphics Production 35 pages + 1 appendix 14 March 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Instructor(s)	Keijo Länsikunnas, Lecturer

The goal of this project was to enhance and develop an application serving the purpose of automating the routine tasks performed by the employees of a company making digital visual effects for movies. The tasks involve various operations with image sequences, videos and special software usage. Reducing the time it takes to complete these regular tasks is the main purpose of the enhanced application.

The application platform was limited to the Linux operating system only and the scope of the project converged to a single company. The tools used in the application development were Python programming language, PyQt4 application framework, bash scripts and special command line utilities. Multiple scripts served as the origin of the application creation, so the main techniques of its development were procedural and object-oriented programming paradigms.

The main enhanced module completed during the project was the automatically generated script for further converting image sequences to videos with appropriate labels and a possibility to change several output parameters. Other implemented functionality included integrated file browser enhancements for remembering local user settings, a convenient in-browser sorting of the sequences, utilities to control permissions on the local server and to power on the render machines remotely. Some graphical user interface improvements were done as well.

The results are significant for the general workflow of the company, since both the enhanced modules and the new application functionality save considerable amounts of time and make routine tasks completion more convenient. Further development of the application may focus on other automation tasks, advancing of the graphical user interface and creation of the unified package for the application installation.

Keywords	automation, Linux, Python, PyQt4, bash

Contents

1 Introduction 1

2 Theoretical Background 3

2.1 Comparison of Studio Manager and Project Management Software 3
2.2 Studio Manager as an Automation Tool 6
2.3 Justifying the Need for Studio Manager Enhancement 9

3 Tools and Methods 13

3.1 Overview of Studio Operating Systems and Software Integration 13
3.2 Outline of Used Programming Paradigms in Relation to Studio Manager 14
3.3 Overview of Tools and Environment for Studio Manager Development 17

4 The Implemented Functionality 21

4.1 The New Daily Script Implementation 21
4.2 File Browser Tab Enhancements 23
4.3 SSH Tab Enhancements 26
4.4 GUI Improvements and Documenting Issues 28

5 Discussion 30

5.1 Overall View of Studio Manager 30
5.2 Benefits and Possible Improvements of Studio Manager 32

6 Conclusions 34

References 35

Appendices

Appendix 1. Key Code Excerpts

1 Introduction

Project management and organization are important aspects of a project development process. Often simple tables, figures, graphs, timetables and scheduled meetings are not sufficient for successful and efficient project organization. Sometimes routine tasks are present within a project, and completing these tasks manually takes an unreasonable amount of time and requires additional resources. Therefore, the special software is needed to fulfil the purposes of a certain type of project.

The goal of this paper is to describe the enhancement and development of a project management application serving a number of purposes, mainly automating the routine tasks of a project. This application is created within a company called Algous Studio. The company has more than ten years of experience in digital graphics production, such as creating visual effects for movies, including 2D image compositing, animation, 3D modelling, lighting, shading and texturing. With the number of customers and orders being constantly increased, it is necessary to quicken and automate the most common tasks of the employees. This was the main reason for starting the development of this application and it is still the main reason of the current project of enhancement and adding new modules to the application. The application is named after its main purpose – Studio Manager. Its second name is Studio Commandor, referring to the classic file browser implemented in it.

Another reason for developing this application is combining the specific software tools used within the company into one compact application, combining all the vital parts of that software together for further data processing. The data mentioned here is common for various projects in visual effects creation – usually image sequences, video files or software specific types of files. In addition, there are certain operating system specific routines that employees have to perform occasionally. Most of these tasks take significantly more time to complete if done manually.

The scope of the project is strictly limited to Algous Studio, not only because of the confidentiality reasons, but also due to the organization of the code of the application itself, written in a way to constantly interact with the studio's local server in most of its functionality implementations. Therefore, it would take a considerable amount of time to

rewrite the code to adapt the application to any other environment. However, since the primary operating system to run Studio Manager is Linux, the general idea of free and open source software applies. Hence some parts of its code could be transferred and used outside the scope of the studio needs. Another issue concerning the scope is the operating system. All the machines in the studio are running different versions of Ubuntu or Ubuntu-like systems such as Linux Mint. Therefore, it is important to outline that the application runs on all of these systems, provided the needed libraries and packages are installed. At the beginning of its development certain functionality of the application was tested on Windows systems, but as more modules were added, Windows support was deprecated.

2 Theoretical Background

This chapter discusses the general idea of project management and describes the similarities and differences between Studio Manager and other project management software. The general workflow of the application is described. Automation software examples are outlined and the close relation of Studio Manager to automation software is shown. The old version of the application is described in this chapter as well, in order to justify the need for the implementation of new modules and therefore justify the purpose of the whole project.

2.1 Comparison of Studio Manager and Project Management Software

One of the most important issues in any project is scheduling. The customer is highly interested in getting the product on the deadline, which may be stated long beforehand. If the deadline is not met there can be consequences, and neither side is interested in them. Looking from the project manager point of view, it is also important to see how the project progresses, how close it is to the actual completion and what are the realistic possibilities of each employee involved. Also, in some cases different projects depend on each other. Moreover, some employees might be working on several projects with different priorities. Keeping all these matters in mind makes project scheduling and tracking vital in the project management process. [1, 5-7]

In Algous Studio the monitoring of any project progress, along with other techniques, can be achieved through so called *dailies*. The dailies are special videos, created by most of the employees regularly. These videos are often put on the server on a daily basis, hence the name. The purpose of dailies is to show a certain modification, or multiple modifications, of a shot received from the customer, so that the project supervisor can view it, give feedback or commentaries, mark it as completely accepted, partially accepted or to be redone. The main difference between Studio Commandor and classic project management software is that a feedback system is not implemented. The feedback is taken care of through another special software tool. Instead, Studio Commandor provides the file organization and the convenience to access dailies created for a given shot of a given project. Figure 1 illustrates an example of a dailies list created for a single shot.

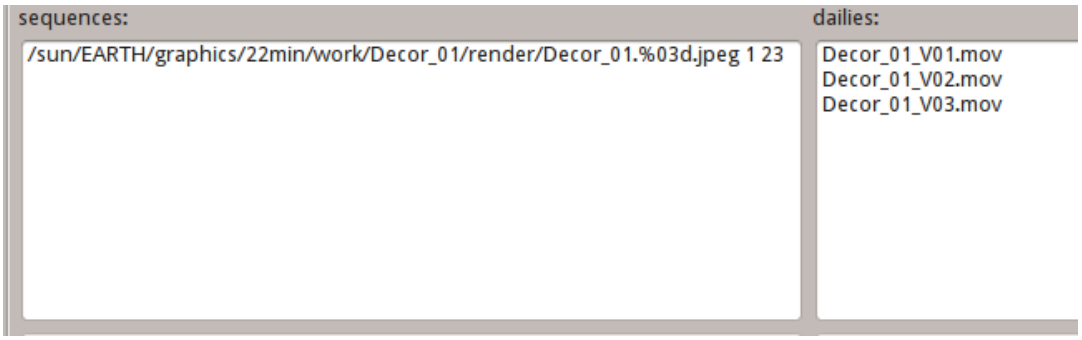

Figure 1. Dailies list of a shot as implemented in Studio Manager.

There are three versions of dailies created for a shot located in a folder, the full path of which is shown in the left part of Figure 1. Dailies themselves are listed on the right side. It is important to note that filenames for dailies are generated automatically from the sequence name of the shot with a version number appended (V01, V02...). Also, the full path of dailies is not displayed, since the application itself takes care of the strictly defined directory system on the server. The newly created folder in the daily path may confuse the user, so the shot sequence path is enough. Besides, this path is given for information only. This system makes the access of dailies easy, prevents the user from creating wrong directories and lets the project supervisors reach any version of dailies at any time.

Another important issue brought up by the system described above is control. If an employee is highly experienced and trusted, the project supervisor might require less reporting from him or her. As the deadline approaches, the employees would want to focus on the actual project rather than the reports. However, this might be dangerous for the project outcome, if the time between reports increases. Therefore, the balance between agility and control is needed in this type of project development and management. Although such balance always depends on a number of factors, including cultural issues or supervisor involvement in the given project, a balanced approach would still be essential. [2, 109]

The type of project management in Algous Studio is a combination of traditional and agile. Naturally, in the inception phase of the project the workflow is steady and unperturbed, since the human resources are highly valued. As the project shifts closer to the deadline, its type also shifts towards agile development, since there is always a risk of uncertainty in understanding between the customer, the project supervisor and the employees involved in the project. While all misunderstandings are usually managed

through personal communication, the technical side of the project still largely depends on the Studio Manager application. This is achieved by controlling the regularity of dailies and their versions for any project. For example, if the customer wants to see a preview of a certain shot that is still in progress, the request is sent to the project supervisor, who in turn requests a preview daily from a person working on the shot. This is done rapidly using Studio Commandor, which makes it a useful tool in both project control and communication with customers. The general workflow of the dailies creation and its control by supervisors is shown on Figure 2.

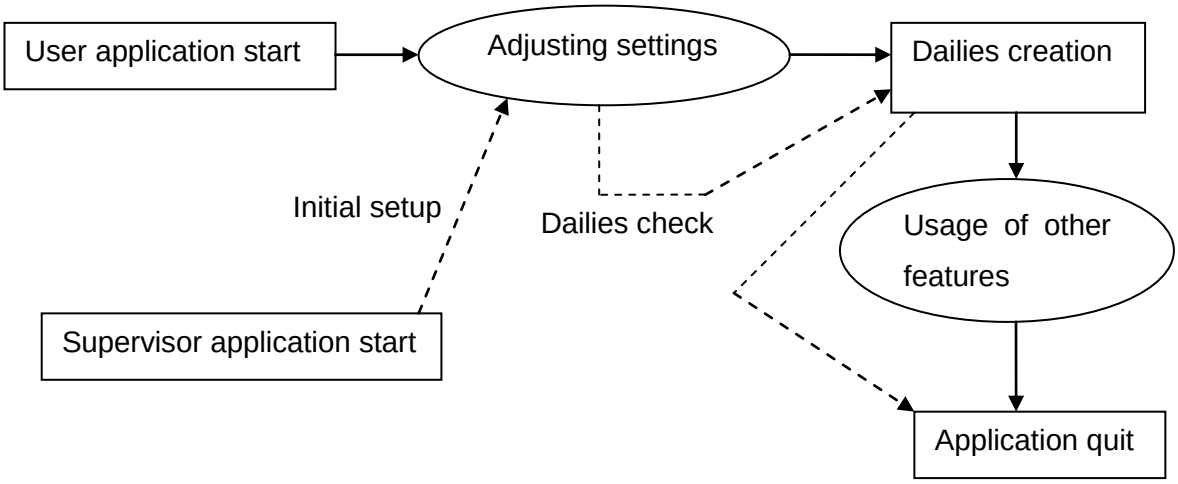

Figure 2. General workflow of dailies creation.

Figure 2 shows the two perspectives of the dailies creation. The solid arrows represent the user perspective on the workflow with elliptical parts being optional. The dashed arrows correspond to the supervisor workflow, which includes the setup of initial settings and checking the created dailies. It can be easily concluded from Figure 2 that the application operation resembles the project management control. However, the application is essentially a technical tool rather than a control tool, since the feedback system and the project stages and milestones are missing.

Considering the facts depicted above, Studio Manager can be presented as project management software to a certain extent only. The possibility of tracking the stages of a project through dailies is a property that firmly connects Studio Manager to project management software family with Microsoft Project as its classic example. On the other hand, lacking such features as a feedback system, multiuser forum-like communication or a higher control scheduling module makes the application quite different.

2.2 Studio Manager as an Automation Tool

Users and developers of the Linux operating system often refer to a slogan “free as in freedom”. This is occasionally confused with being referred to the price of the operating system, which is distributed free of charge. However, the true meaning of the slogan lies in the freedom of power to know in detail how the computer operates and which tools it is using. There should be the freedom to choose these tools and their combinations in order to take the most of the computational power of the hardware. An important part of any Linux distribution is the command line. Although its user interface is rather far from being friendly, there are undeniable advantages of taking control over the system and its components with further automation. Mastering the command line broadens to creating various scripts that allow completing rather complicated tasks automatically. [3, 26-27]

If such tasks are required to be completed regularly, the need for an automation tool becomes urgent. Time is always a valuable resource that should be preserved, if there is a possibility of automating routine tasks. A script written using a high-level programming language or a shell script containing a set of commands are good examples of automation tools. As the task becomes more complex or the number of routine tasks increases, a script may not be enough to execute the task flawlessly. A change in several parameters can still be handled by the script, but if the parameters affect the inner algorithms of the program flow, an application is the choice. One example of the mentioned type of an application is KRename, the batch rename program for KDE (K Desktop Environment), a popular Linux desktop environment. One of its main tabs is illustrated in Figure 3.

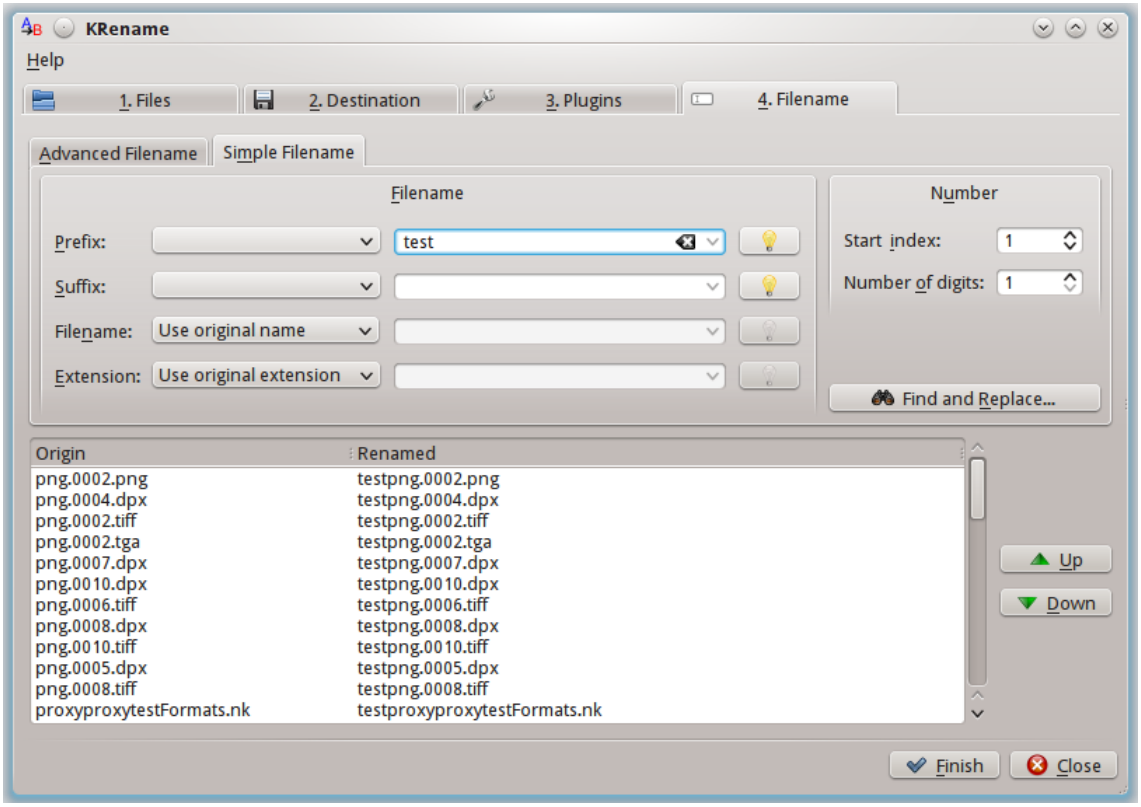

Figure 3. Filename tab of KRename.

The primary and only purpose of KRename is file renaming. However, as Figure 3 shows, the program has multiple tabs, buttons, combo boxes and other elements of a graphical user interface. The reason for that is the presence of numerous parameters that users can configure before the actual renaming takes place. A complex algorithm of renaming a large number of files can be entered by the user to be executed by the KRename application. Therefore, one might say that file renaming is automated by KRename on a high level. Definitely, there are other ways to rename multiple files. For example, a command line can be used for that purpose. The drawbacks of this method are obvious: the user would have to know the exact syntax of the command, specifying all the parameters and still might lack the desired file naming patterns at the output. When using the application, the output patterns are shown before the renaming takes place, as can be seen in Figure 3.

The property described above applies to Studio Manager as well. There are a few tasks within the studio that are simple in essence, but require a considerable amount of time if not automated. One example is software installation. If a certain package is required to be installed on ten computers, manual installation would take time. Since the

action is repeated, there is a way to automate it. In Studio Manager this feature is implemented in the SSH (Secure Shell) tab. Figure 4 gives a visual representation of the mentioned feature.

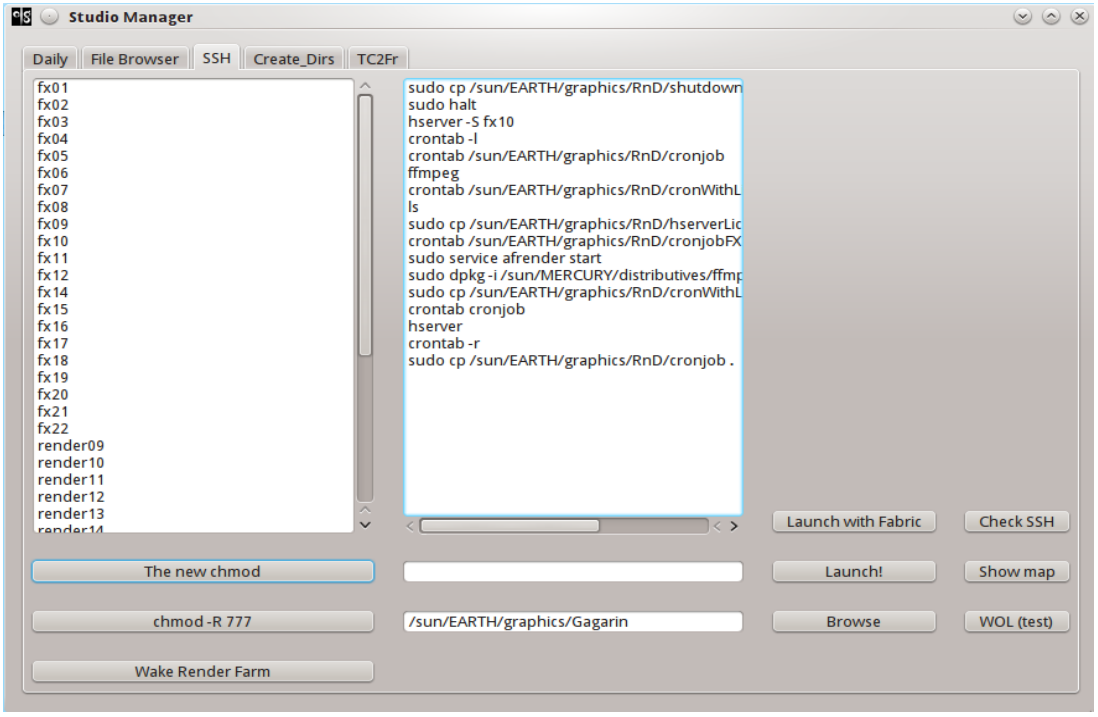

Figure 4. Studio Manager SSH tab.

There are two lists in the SSH tab of the application, as illustrated by Figure 4. The list on the left contains the hostnames of the workstations in the studio, including render machines. The list on the right is a history of commands that have been executed at least once. By selecting the necessary hostnames from the list on the left and an appropriate command from the list on the right, it is possible to launch this command to the selected hostnames, provided the SSH connection is up. It is also possible to type a new command, which will be stored in the history list. Typical usage of this feature applies to installation of new software, starting special services or copying files, but the application allows other standard Ubuntu bash commands to be executed as well. The idea of freedom to choose and combine software and the idea of automation converge in this particular feature. The powerful command line hidden from the user and a graphical user interface with intuitively clear lists and buttons are working together in an essential feature for automating trivial tasks.

Another example is the *chmod –R 777* button, which can be found in Figure 4. Occasionally the employees need to process files through a graphics program into a specific directory. Typically, these files are image sequences. However, if the render machines are involved in the file processing, the directory might not have the full permissions to accept all the files generated by the render machines. The workaround for a user is to set the right permissions for the directory. The *chmod –R 777* button deals with it in seconds by sending a bash command that gives the full permissions to the specified folder. The same procedure would require a manual SSH connection to the server, typing the password, locating the directory and finally issuing the command. Naturally, automation saves a considerable amount of time here as well.

Although Studio Commandor is a multi-purpose application, the majority of its features deal with reduction of time of the tasks completion by automation. Dailies generation, described in section 2.1, is an automated task. **Create_Dirs** tab of the application is responsible for creating a proper directory tree for each set of shots within a project. There are about 10 subfolders being created on a single button hit. The *Wake Render Farm* button located on the SSH tab powers on the render machines, ‘waking’ them from the shutdown state. All of the outlined examples point to Studio Commandor as an automation tool, which is its primary software type. It is important to emphasize the encapsulation of the features related to the command line. The inner mechanisms of the application use the benefits of the shell, but it is hardly relevant for the user to know the details of these mechanisms. In the case of Algous Studio the user is primarily interested in the end result rather than the tools involved. Thus, Studio Commandor is being designed as an application that combines various software implementations, but the exact combination techniques are concealed.

2.3 Justifying the Need for Studio Manager Enhancement

The first reason for the development of any software is its relevance. For an application with a strictly limited scope it is especially important. When the old version of Studio Manager was in use, the data on the number of used features among the studio employees was gathered. A single feature is one implemented functionality unit, such as creating dailies or creating a tree of directories with the help of the application. Table 1 presents the gathered data.

Table 1. Intensity of the usage of Studio Manager features before enhancement.

Regular usage of Studio Manager features	**Number of employees**
No regular usage	2
1 feature	4
2 or more features	7
All features	2
Desire for enhancing existing features	4
Desire for implementation of new features	5

With a total number of employees summing up to 15, Table 1 shows that the majority of employees at Algous Studio use at least 2 features of the application regularly. More than half of the employees also desire for either enhancing of existing features or the implementation of completely new features they think would save time for the completion of various tasks. Thus, the data presented in Table 1 justifies both the relevance of Studio Manager and the need for its enhancement.

A graphical user interface is an important aspect of an application, since its design may determine the usability of the application. It has to present the information to the user accurately and clearly. At the same time the user has to understand the actions available to him or her and the possible outcomes of the performed actions. Usually a graphical user interface designer is confronted with a challenge to compromise the appealing interface and a possibility to interact with a user in a way to improve the effectiveness of the jobs processed by the software. Such a compromise should be taken into account. [4, 103-104]

Since functionality was the number one priority as the Studio Manager development started, relatively small effort was taken to design a proper user interface for it. At the inception stage of the project the proper implementation of each button's function meant significantly more than its appearance on the application window. Hence the old version of Studio Manager had a poorly developed graphical user interface, the examples of which can be seen in Figure 5.

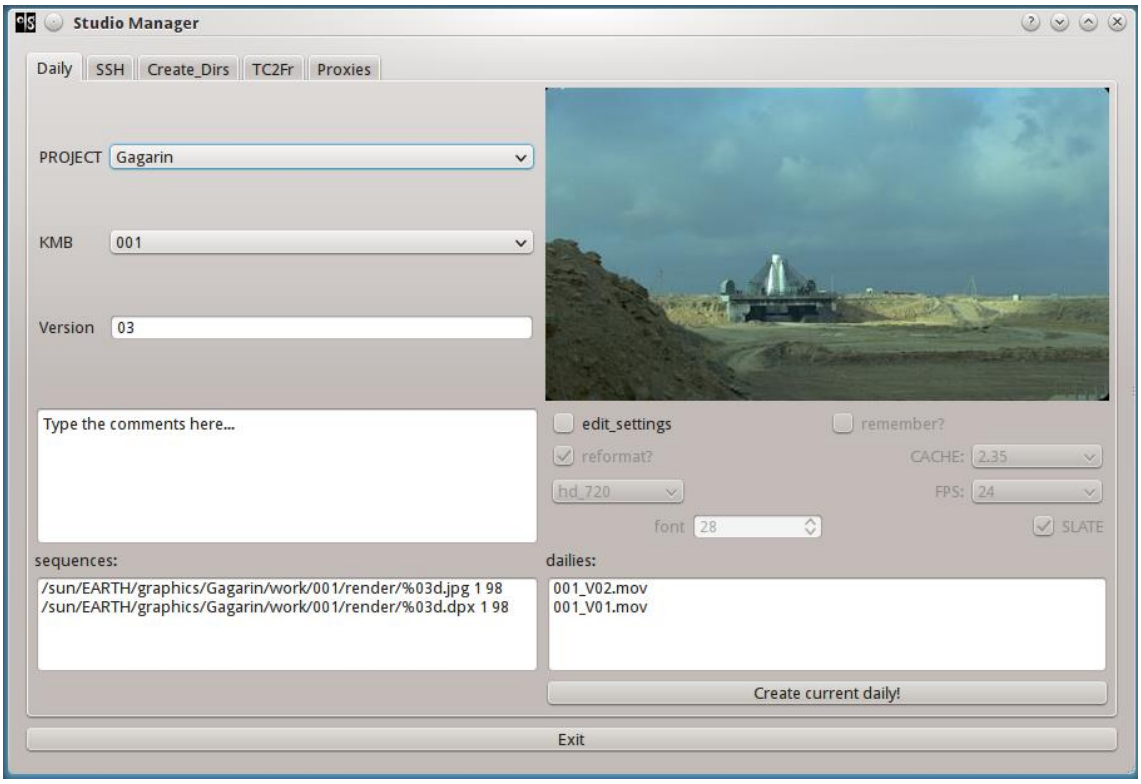

Figure 5. Main tab of the old version of Studio Manager.

The options for dailies to be created are displayed on the right side of Figure 5. It is hard to say how the options are related to each other or to the dailies list below, since the location of the options' widgets is disordered. For a new user the handling of these options would be challenging, mostly due to the weakly implemented user interface. There is a snapshot of a sequence above the settings section, but none of the sequences are selected. In fact it is the last frame of the last sequence in the sequence list. This is exceedingly inconvenient and may as well be confusing for users, since it is possible to have multiple different sequences within a shot and showing the last frame of the last sequence with none selected would be absolutely incorrect. Another example is the *Exit* button located below the sequences and dailies lists. The user might not understand the purpose of this button, since there is a default close button in the upper right corner of the window. The exit button was implemented for saving the window size and location on the screen for the next launch of the application, whereas the default close button would not save the mentioned settings. However, the user is not informed of such nuances of the application nor is it intuitively clear.

By the facts outlined above the need for Studio Manager enhancement and further development is justified. There are two main reasons to widen the application's capabili-

ties. One is the relevance of the application within the studio. While the need for implementing new features or enhancing the old ones is desired by the users, the importance of the application development would be significant. Another reason is the need for creating a more comprehensible graphical user interface, which would appeal to the users and make general application handling more straightforward.

3 Tools and Methods

This chapter describes the software environment of a typical workstation of the studio and explains the idea of transition from a number of scripts to a single application with multiple modules. The ideas of object-oriented programming and procedural programming as the main programming paradigms of the project are introduced. The Python programming language as well as PyQt4 binding for a well-known graphical toolkit Qt are introduced as the main tools for Studio Commandor development. The Eclipse environment and the project testing methods are outlined in order to explain the details of the application development progress and justify the type of software development as agile.

3.1 Overview of Studio Operating Systems and Software Integration

When choosing from numerous Linux distributions, one should think of the purposes the operating system would help him or her to achieve. Although these distributions support similar philosophies, such as “free as in freedom”, the approach to the end user might be different. Ubuntu is occasionally called “Linux for People”, meaning that it is aimed at being user-friendly. People who wish to get their job done regardless of the operating system trivia choose Ubuntu because of its comparative simplicity. For example, installing additional software could be done through Synaptic Package Manager, containing a wide range of packages for different needs. Moreover, the operating system itself includes useful packages available right after the installation “out of the box”. The Ubuntu community is vast and devoted, providing support for end users in various matters. It is also a rapidly developing operating system with regular updates coming from official repositories. [5, 13]

For most of the artists within the studio it is essential to keep the system stable and simple. Thus, Ubuntu is a suitable Linux distribution for workstations in the company. Normally, an artist with little engineering experience would need specific software and a couple of system tweaks only. Such tweaks can usually be done within a clear and user-friendly Ubuntu interface. In fact, the end user will hardly ever need to use the command line, although it is always there for more sophisticated purposes, described in Chapter 2. With updates coming regularly, it is possible to keep the system stable and especially convenient for studio artists, keeping the opportunity to alter the system as well.

On the other hand, not all software is available in the official Ubuntu repositories, nor it is obtainable from additional ones. One example is Afanasy – a powerful open-source render farm manager. Like other software of this type, it should be downloaded separately. The need for explicit compilation and configuration makes Afanasy's installation even more complicated. However, Linux systems allow most of the software to work just by copying the directory with previously compiled and configured files, in case the creation of the package is not possible. Adding several libraries and setting a few environment variables on the local machine might be required, but still this kind of installation would save time.

The mechanism described above resembles the Studio Manager application launching system, although the latter is simpler. The application itself is located on the server. All the user needs is a link or a launcher of the application file. Naturally, a connection to the server is required. In addition, the packages and libraries that are necessary for Studio Manager to run should be installed. Nevertheless, the Ubuntu operating system provides enough suitable tools for such purposes and makes the integration of required studio software quick and convenient. It is important to note that the outlined integration method applies to the Studio Manager application as well.

Although Ubuntu is the main system used in Algous Studio, two facts should be kept in mind. Firstly, there are different release versions of Ubuntu. Secondly, there might be other Linux distributions installed for advanced users. Luckily, the main tools used for the development of Studio Manager – Python, PyQt4 and of course bash shell – are common for most of the distributions. Thus, application development and usage would be easily expanded to any Linux system, which slightly broadens the scope of the project.

3.2 Outline of Used Programming Paradigms in Relation to Studio Manager

The history of Studio Manager started with several automation scripts, written in either Python or bash. Eventually, the number of scripts increased, since more routine tasks needed to be automated. In addition, the complexity of the scripts varied according to the task complexity. Maintaining these scripts became more demanding. Moreover, the script usage became problematic for users, since they had to be launched separately. In case of errors the user could have spent time finding out what exactly went wrong.

At the point when scripts maintenance became too complex and their usage too inconvenient, the actual application development started. It was logical to divide the main application window into tabs, representing modules that contained closely related implemented functionality. Although the application implying a graphical user interface usually excludes the usage of the console, it is possible and often recommended to launch it through the command line in order to track the important runtime information and error messages. It helps the further debugging and lets the user follow the program flow.

While various graphical toolkits, including PyQt4, with a variety of widgets and buttons involve object-oriented programming style by default, it is still important to outline its benefits. This approach combines both structured logic and structured data, providing easy code factoring and reusing. Encapsulation makes the code safe and not too challenging to debug. Keeping the class methods organized with the help of the object-oriented paradigm helps making the entire project more stable on the long-term scale. It is also natural for people to think in objects rather than abstracted variables. [6, 26-28]

Apart from the default classes embedded in PyQt4, which are mostly part of the graphical user interface, such as buttons, text fields and combo boxes, there are more complex classes in Studio Manager. One example is a browser tab class, which is mainly responsible for creating the file tables of the current directory. Two browser tabs take most of the space of the file browser tab of the application, as illustrated by Figure 6.

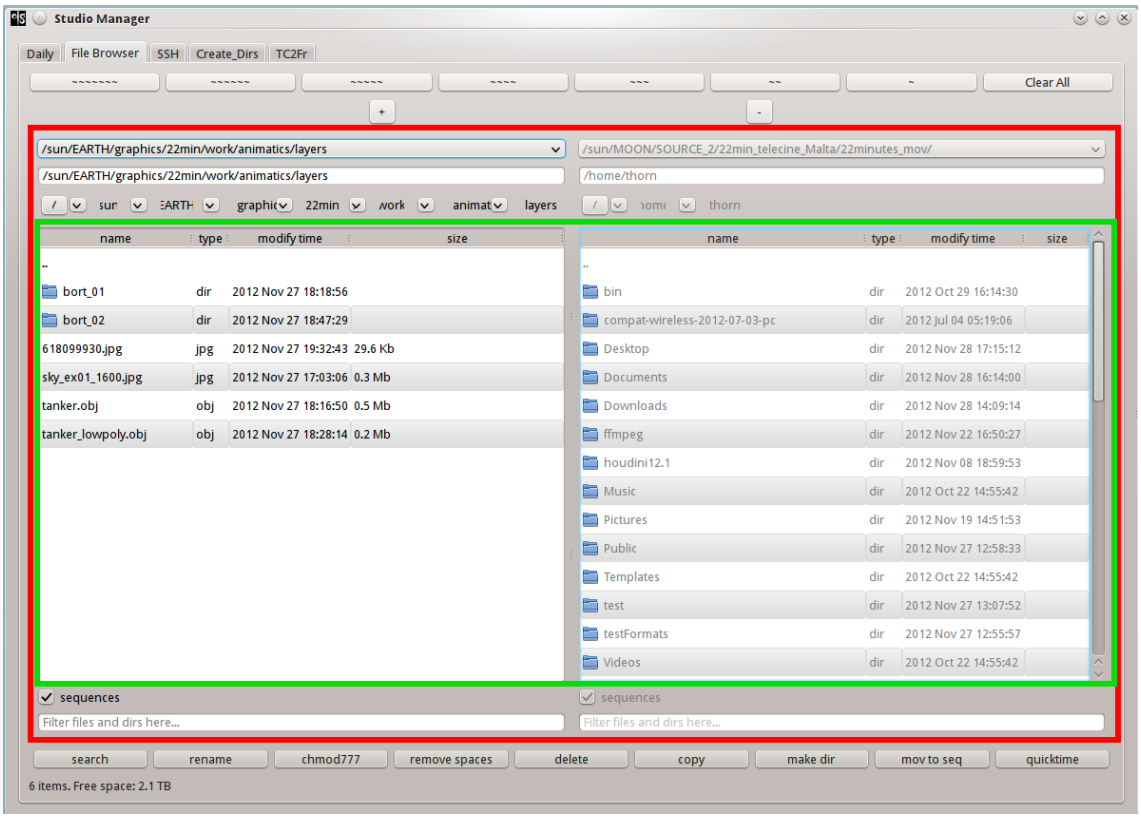

Figure 6. Browser tabs embedded into the file browser of Studio Manager.

The red rectangle encloses the area where two browser tab objects are located. It can be clearly seen that these objects are similar but have different parameters. In this case the addresses differ, so different file tables are shown. Although the traditional file browsers have only two file tabs, there is a possibility to add more by creating another object of the same class. In turn, file tables themselves are objects of the same class called a browse table. Two browse table objects are enclosed by the green rectangle, as it is demonstrated by Figure 6. This is a vivid example of utilizing the object-oriented programming style, specifically reusing the code. If the implementation of another browser tab was needed, it could be easily added by creating an object of the corresponding class, which would contain the needed attributes by default.

Although object-oriented approach is more appropriate for the Studio Manager application, as outlined above, the application's origin is a collection of scripts, which implies a different approach, namely procedural programming. Since automation tasks may not always be very complex, the procedural approach still applies. Often it becomes inappropriate solely in tasks whose implementation reaches a high level of complexity. Oppositely, the unchallenging tasks solutions could be implemented as just a sequence of

instructions. These solutions would be highly effective for the mentioned type of tasks. Furthermore, a procedural approach is the most natural way of programming for less experienced developers, since it involves defining the problem with further dividing it into smaller parts. Often object definitions and encapsulation as well as the other characteristics of OOP are not entailed by the task. In such cases the procedural approach would satisfy the developer's needs. Nevertheless, the Python programming language allows relatively easy rewriting of the code using OOP, if it is strictly required. [7]

Summarizing the details described above, it is essential to note that Studio Manager development makes use of two programming paradigms: object-oriented and procedural. The first one suggests encapsulation, code reusing and general application stability. The second one adds the possibility to maintain the simpler tasks support and implementation while keeping the object-oriented trivia away and thus making the code less complicated. Combining the benefits of these approaches makes Studio Manager development and enhancement rigid and safe, yet more comprehensible for a programmer.

3.3 Overview of Tools and Environment for Studio Manager Development

The main programming language used for the development of Studio Manager is Python. There are several reasons for this choice, apart from Python being available in most of the Linux distributions, including Ubuntu, by default. Firstly, Python supports system programming, meaning that it is easily compliant with system utilities such as shell scripts and the command line itself. The built-in module 'os' makes the use of system tools, such as directory files processing and SSH connections more accessible than the syntactically obscure shell languages. At the same time Python allows keeping code clarity. [6, 73]

Secondly, Studio Manager needs a controllable and efficient graphical user interface. Python's simple syntax and a possibility to utilize the object-oriented approach comply well with the GUI model. Indeed, it is natural to represent the various devices drawn on a screen as Python classes. Furthermore, Python allows making changes to the GUI layout and observing their effects rapidly, giving the developer a good opportunity to experiment with alternative designs and program behaviour. [6, 357]

Thirdly, it is important to contrast Python with compiled languages such as C and C++. Although the latter two are classic examples of the procedural and object-oriented ap-

proach respectively, these languages are optimized for speed of execution at the cost of a higher complexity. Python, on the contrary, is optimized for speed of development at the cost of program performance. Nevertheless, Python has a useful characteristic of integrating with components written in C. In case a performance increase is needed for a certain program module, it is possible to merge the Python code with the C code to achieve that particular purpose. [6, 1483]

The majority of workstations at Algous Studio have powerful high-end CPUs and a sufficient amount of RAM in order to perform rendering, lighting, shading and other demanding tasks that require high computational power. Therefore, for comparatively small applications such as Studio Manager execution optimization would not bring significant positive results. Moreover, proper and efficient task completion has priority over the minor runtime delays, which makes Python more suitable for this particular application development than C or C++. It is also essential to briefly point out that the Python version 2.7 was chosen, since it is currently stable and available in most of the Linux distributions by default.

PyQt is a binding of Python and Qt, meaning that it makes Qt application framework libraries available to Python programmers. Qt, in turn, is a powerful toolkit with numerous features. One is Unicode support, which makes translation and localisation as simple as possible. This is especially important in environments where English is not the only language used. Secondly, Qt provides the signals and slots mechanism. This innovation, introduced by Qt, provides the abstraction of event handling. It grants the high-level view of the events, which makes handling the specific actions, such as document saving, undemanding, even if the same action is invoked in different ways. Moreover, this mechanism ensures type-safe communication between objects when emitting or receiving of signals occurs, because the objects do not have any knowledge of each other. The knowledge of the signal being sent is sufficient. [8, 5]

PyQt4 combines the benefits of the Python programming language, outlined earlier in this subsection, with the benefits of Qt framework. Nearly all of the Qt classes are available to Python programmers through PyQt. Naturally, all of the highlights of Qt mentioned earlier are accessible in PyQt as well. Since all of these highlights are crucial for agile development of Studio Manager, the usage of these tools is justified. [8, 7]

Apart from the programming language and the toolkit for graphical user interface development, it is vital to describe and justify the integrated development environment. Eclipse IDE was chosen for Studio Manager enhancement due to its clear interface and fast operation. A snapshot of Eclipse Juno with PyDev plug-in installed is illustrated in Figure 7. The Eclipse window has been shrunk for convenience.

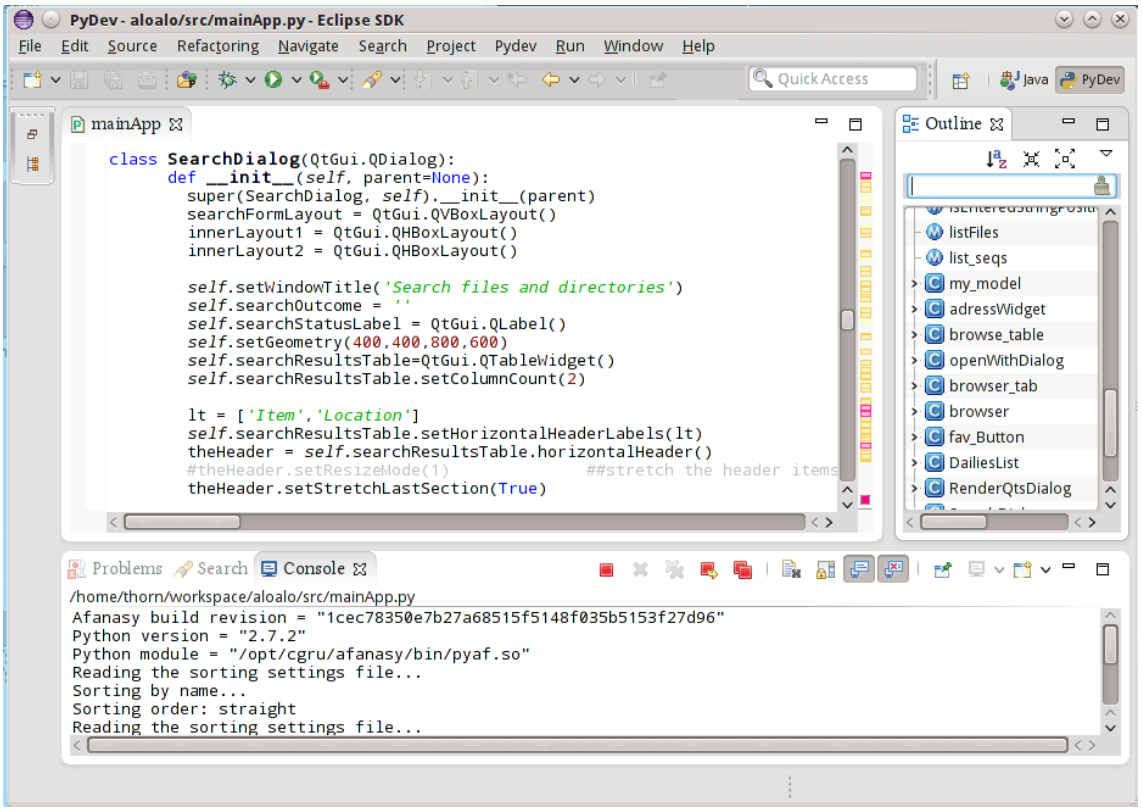

Figure 7. A snapshot of Eclipse IDE with PyDev plug-in in action.

As Figure 7 shows, this integrated development environment contains all the necessary sections. The sections include main menu panels on top, the outline of classes and methods on the right and the console output at the bottom with a possibility to switch to *Search* or *Problems* tabs for searching for specific variables or methods and tracing bugs respectively. The project tree on the left has been shrunk to illustrate the opportunity to control the workspace area by minimizing the sections. Although the program interface might seem complex, it only contains the sections and menus that are used more frequently. Other settings can be reached and altered by finding submenus, which makes Eclipse's GUI both appealing and convenient to use.

Although Java is the default language for Eclipse, plug-ins for the support of other languages are easily installable. In the case of Studio Manager, the PyDev plug-in is used. As its name suggests, this plug-in is responsible for integrating the Python interpreter and code editor into Eclipse. In the upper right corner of a window snapshot the PyDev perspective is opened, as Figure 7 shows. It is crucial to note that code highlighting is available by default, which makes the coding process more comfortable and reduces the probability of typing errors. On the right to the main section there is a vertical ruler indicating warning and errors, which makes the in-code navigation simpler. These advantages justify Eclipse IDE as the suitable environment for rapid and effective development of Studio Manager.

It is also essential to mention that the application testers are the studio employees using Studio Manager. Once a bug is noticed, it will reported and further reproduced by the developer. Reproducing the bug takes the minimum time, because the Eclipse program launches the application rapidly – usually a few seconds is enough. If necessary, the debugging feature can be used as well. The bug is then fixed and the new version of the application is pushed to the local server. Occasionally the users ask for either short-term or long-term enhancements, depending on their urgency and complexity. These wishes for enhancements are recorded for further implementation.

By the facts presented above it is obvious to state that the application development style is definitely shifted to agile. Indeed, all the outlined tools and methods of the project, including the Python programming language, the PyQt toolkit and a quick responsive approach to the testing signify the agile software development. This method is the most appropriate for Studio Manager, since the functionality of its modules and the actual task completion are the priority before the functionality achieving methods and rationalizing the computational power management. That is, altering the basic algorithms for a minor performance improvement is not needed in case the specific module is capable of its task accomplishment.

4 The Implemented Functionality

This chapter describes the main enhancements of Studio Commandor that were implemented during the project. Firstly, the new script for creating dailies, which works significantly faster, is introduced and compared to the previous one. Secondly, the file browser enhancements are described. Thirdly, the new SSH tab functionality is outlined. Finally, the graphical user interface improvements and code documenting issues are covered.

Essentially, the enhancements of the application can be divided into two major groups. The first one is performance improvement, used in the new daily script implementation, which is described in detail in section 4.1. Since the technique used to generate dailies was ineffective, the new technique had to be implemented to boost the performance. The second one includes various functionalities implemented through GUI improvements. The latter relates to the file browser tab and SSH tab enhancements described in detail in sections 4.2 and 4.3 respectively, as well as the general GUI improvements outlined in section 4.4.

4.1 The New Daily Script Implementation

As stated in Chapter 2, the creation of dailies is vital for tracking and evaluating the work progress in Algous Studio. Since dailies are created on a regular basis and the employees always have to check the result, the time spent on their creation should be reduced to a minimum. The older version of the dailies script used the Nuke software to generate frames for each daily and then combine them into a single video container. The video should be of a given size, selected from the available size formats. There has to be multiple labels on the video itself, providing information on the current project, the current shot, daily version, the date and the frame number. Additionally, there should be a so called **cache** present. The cache is responsible for partial hiding of the top and the bottom of the video and its intensity can be chosen from three alternatives. Optionally, there should be a **slate frame** coming first and displaying the same information on a black background.

The old technique of creating the frames for a daily involved reworking each frame separately and putting them into a separate folder as temporary files for further combin-

ing them into a daily using **ffmpeg** – a powerful tool for video manipulation. Although a single frame processing usually takes less than half a second, the daily creation process can still take a significant amount of time if there are hundreds of frames in a shot. In addition, the only information label that is changing throughout the video is the frame number. Therefore, processing every image of a sequence is not needed. The new dailies script uses a technique of putting the readymade labels on a video, which is combined of the existing sequence images, so that multiple image processing is avoided.

The tools used for implementing the described new technique are **Imagemagick** – a tool for image manipulation and the mentioned ffmpeg. First of all, the general information on the sequence is gathered. It includes the project name, the name of the shot, the current date, and the optional notes to the daily that is going to be created. Then Imagemagick takes care of finding the image resolution of the given sequence. The number of the first frame and the sequence duration are calculated at this point as well. After that the daily options are read from the main Studio Manager tab. Finally, the new daily script generation begins, where all of the options are taken into account.

The daily script is a regular executable bash script consisting mostly of Imagemagick commands. If the slate frame option is checked, it is created with all the needed labels included and put into the same folder with the sequence as a preceding frame. Then the canvas of the needed size with a cache and its own labels is created. The last command of the script is utilizing ffmpeg to create a video from the given sequence with a canvas over the video. In addition, ffmpeg is responsible for numbering the frames. In the end of the daily creation method the slate frame is deleted and the newly created daily is opened for viewing in an open-source sequence viewer djv_view. The full code of the *create_daily* method, including the preliminary settings and the script generation, can be found in Appendix 1.

It is important to note that internal script generation is a unique case in the application. The reason for it is the intensive usage of other software, namely ffmpeg and Imagemagick. Such usage makes bash scripting more convenient than Python code implementation. Nevertheless, the script is not meant to be modified or executed outside the application, which would contravene the general idea of automation.

As outlined, the new technique avoids processing of every single frame of the sequence, overlaying a canvas with labels on the generated video instead. It is essential to emphasize the main advantage of the new script compared to the old one, which used the Nuke software. This advantage is definitely saving time. The data on the daily creation time from the sequences of different duration using both scripts is gathered in Table 2.

Table 2. Old and new script daily creation time dependency on sequence duration.

Sequence duration, frames	**Processing time, s**	
	Old script	**New script**
23	5	2
45	9	4
98	22	8
389	83	21
459	97	23
600	124	27

As demonstrated by Table 2, the new script is able to create dailies faster than the old one from the low duration sequences already. However, as the sequence duration increases, the new script becomes much more efficient. It should be emphasized that the old script processing time grows more or less linearly with the growing sequence duration, whereas the new script processing time does not increase that rapidly. The old script behaviour is easily explained by reprocessing each frame. Conversely, the new script focuses on creating the canvas once with further hasty video conversion by ffmpeg. The new technique is a vital implementation in Studio Commandor, since it makes the dailies creation significantly faster, especially if the high duration sequences need to be processed.

4.2 File Browser Tab Enhancements

There are several enhancements that were added to the file browser tab during the project. It is appropriate to present them visually with the help of Figure 6. The topmost row is populated with buttons that accept a drop action if an existing location is dragged

onto them. In other words, it is possible to drag a directory to the button and it will store its location, which can be further accessed by pressing that particular button. These locations are saved in a file, so that the buttons are restored after reopening of the application. The '+' and the '-' button positioned below allow adding and removing temporary bookmarks. Basically, the operation is the same, but the number of buttons is unlimited and they are deleted once the application is closed. The combo box below is a stack, populated with 10 different locations that the user has visited last. These locations are also stored in a file, so that it is possible for the user to switch to one of the latest visited locations.

Apart from the location remembering widgets, there is a sequence check box below the file list that allows switching between a normal file list view and a sequence file list view. Also, there is a filtering text box for immediate filtering of the file list if the text is entered. It is important to note the significance of the sequence check box, whose operation is illustrated in Figure 8.

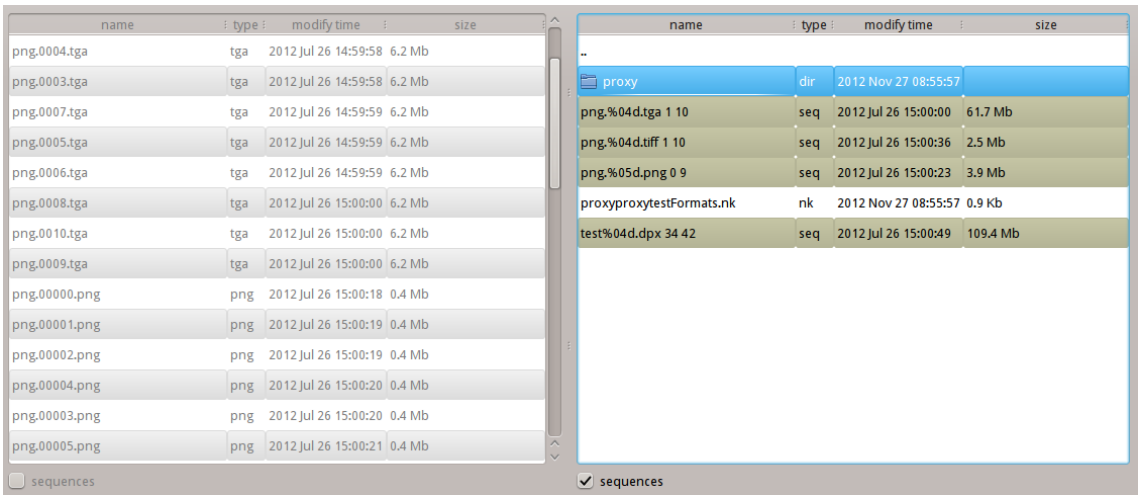

Figure 8. Sequence check box operation.

Both file lists shown in Figure 8 represent the same location in the file system. However, the file list on the left represents separate files, whereas the file list on the right gathers the appropriate files into sequences, since the check box is checked. This is important for the studio, since the whole image sequences are often copied, moved or deleted and the data could be lost if multiple files are processed. It is especially vital for directories, where several sequences are located. The right part of Figure 8 illustrates how the sequence check box alters the file list containing sequences, so that various file operations can be performed easily without the fear of losing data.

The bottom row of buttons, which can be found in Figure 6, contains the new buttons implemented during the project. They are: *search*, responsible for simple file searching; *rename*, used to rename both regular files and folders and sequences; *remove spaces*, responsible for recursive renaming of the files and folders containing spaces, since the djv_view sequence viewer cannot resolve the names containing spaces; *mov to seq*, taking care of opening the mov container and converting a video into an image sequence with the help of mplayer software. An information label showing the number of objects in the current directory, and available free space on the current file system is located below the bottom button row. All of the relevant implementations of the described functions are shown in Appendix 1. The code for constant buttons that are being remembered is omitted due to its similarity to temporary tabs code. Most of the search conditions are also omitted for the same reasons.

Since the work within Algous Studio is often related to image sequences manipulation, the most frequently used button for the bottom button row is *rename*. Keeping the sequence names consistent is essential for proper organization of files. Therefore, the implementation of the sequence renaming should be emphasized. The dialog for sequence renaming is shown in Figure 9.

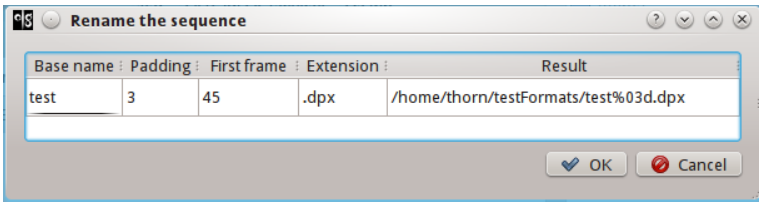

Figure 9. Sequence renaming dialog.

As Figure 9 demonstrates, the user can edit all parts of the sequence name: base name, padding, first frame and extension. The latter changes the format of the images by means of the integrated system tools and should be applied with a high level of concern. The result section of the dialog is showing the full path of the renamed sequence, but is currently scheduled for deprecation from the application, since the information it provides is irrelevant.

Generally, all of the described file browser enhancements are vital for speeding up the routine operations within the studio. Location remembering widgets help the user navigate in the file system rapidly and easily. The sequence check box provides convenient

switching between the file view modes. The filtering text box allows quick file searching within the given directory. The sequence renaming implementation grants the appropriate file organization and ensures avoiding of data loss. Obviously, all the mentioned operations could be achieved separately by utilizing various software solutions or the command line. However, Studio Commandor aims at combining the specific software with specific studio needs for effective task completion and the file browser tab of the application successfully fulfils this aim.

4.3 SSH Tab Enhancements

One of the SSH tab features, the *chmod –R 777* button, was described in Chapter 2 as a very important tool of changing the server permissions. Although the button's operation is fast and simple enough, its drawbacks are obvious. Firstly, the repeated usage of the button is often needed if the new directories are created on the server. Secondly, giving the full permissions to the directories raises the risks of security issues, since the files could be deleted unintentionally. *The new chmod* button, which can be found in Figure 4, is the first step to get rid of the mentioned issues.

The button internal implementation is based on the *chmod –R 777* button. However, the general idea behind the new button is to give the appropriate permissions to a single group, the **users**, which is the default group in Linux. Further on, when an employee belonging to the group needs access to the server directory with new permissions, he or she has no trouble altering the files within this directory. In addition, the render machines that might save the output to this directory are running the rendering processes under a certain user that also belongs to the group. This solves the security issues in most cases, since only certain employees would have access to certain directories. *The new chmod* button is tested for the **users** group only, but more implementations could be added to extend the permissions structure of the server if needed.

It is essential to note that *the new chmod* button also solves the issue of repeated usage, since it forces the newly created files and folders within the operated directory to belong to the **users** group. The bash command behind the button operation is as follows: chown –R :users <directory> && chmod –R g+r, g+w, g+s <directory>. First the command changes the ownership for the given directory and then gives the full permissions for this specific group only. Both operations have the recursive effect on the subfolders and the 's' flag ensures the further files and folders ownership to be under the

users group. This new button significantly broadens the capabilities of the *chmod –R 777* button implemented earlier. Therefore, the old button is scheduled for deprecation.

Another example of the effective command line utilization is the *Wake Render Farm* button of the SSH tab, which can also be found in Figure 4. As the name suggests, the button sends wake-on-lan packets to every machine in the render farm. The render farm is used to process network renders, launched by special software within the studio. Naturally, a considerable amount of work had to be done prior to the button implementation, since the render machines had to be set for receiving wake-on-lan packets. Also, a special script for hourly checking of the processor activity was distributed to the render machines. In case most of the processor capacity is idle, the computer will be switched off.

Before the *Wake Render Farm* button implementation the render machines were switched on and off manually. Obviously, it caused inconvenience and electrical energy waste when the machines were up without any render jobs assigned. The *Wake Render Farm* button provides switching on the whole render farm (currently 19 machines) by a single click. Once a machine is set up to receive wake-on-lan packets, its unique hardware address of a network card could be added to the list, so that the button would affect the new machine as well. Currently powering the machines on is the only automation feature connecting the render farm and Studio Manager. More features, such as sending a specific task to the render machines through the application, are being designed. The frames that are created during this specific rendering task could further be redirected to a certain folder by means of automation. Additionally, these frames could act as a base sequence for dailies creation and thus connect the implementation of the render task management with another Studio Manager module internally.

Just as the new file browser tab features, the SSH tab enhancements speed up the general workflow of the studio. By using the powerful console commands it is possible to set up the appropriate file and directory permissions and preserve these permissions for subfolders. The command line also allows powering on the numerous local machines by sending special packets. From the user perspective all of this is done by clicking the appropriate button, which corresponds to automation as the primary purpose of Studio Commandor.

4.4 GUI Improvements and Documenting Issues

In Chapter 2 it was mentioned that the old version of Studio Commandor had several graphical user interface issues that confused the user. Some of these issues were solved during the project. Figure 10 illustrates the main tab of the new version of the application.

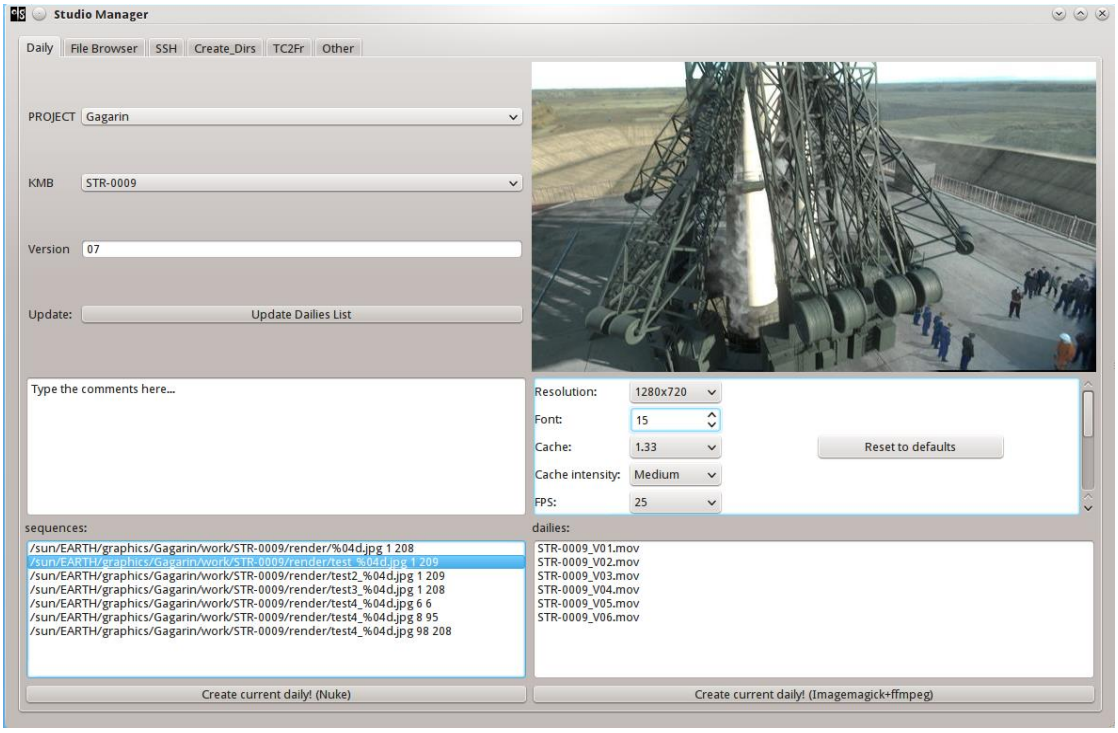

Figure 10. Main tab of the new version of Studio Manager.

Figure 10 demonstrates the GUI enhancements of the application main tab. If compared to Figure 5, the differences can be clearly seen. First of all, the daily settings are placed vertically in a table widget with a scroll possibility. The settings area was completely reworked in order to keep the GUI clean and appealing to users. *Reset to defaults* button was added to change all of the settings to their default values. Secondly, the upper right corner snapshot now actually shows the middle frame of the selected sequence, which is far more comprehensible by users than the previous implementation. Thirdly, the *Exit* button is deprecated and the default closing button is now responsible for saving the window geometry and its position on the screen to be restored on the next application launch.

It is also essential to note the adding of *Update Dailies List* button to the main tab. It is used for updating the current dailies list in case the new dailies were added on the remote computer. The new button in the bottom left corner of Figure 10 is responsible for creating dailies using the old script, but is scheduled for deprecation, since the new script is operational. As outlined, all graphical user interface improvements implemented in Studio Commandor during the project are aimed not only to place widgets on the screen more effectively, but also to be more appealing and less confusing for the studio employees.

In addition to GUI improvements one minor issue should be covered, namely documenting the code of the project. Although it is not an implemented function itself, it closely relates to all implemented functionality described in this chapter. Most importantly, there is no official documentation, but in-code commenting only. Examples of such commenting can be found in Appendix 1. The reason for lack of proper documentation is that initially multiple scripts were merged into an application and further development went on rapidly, maintaining the agile style. Naturally, the small scripts were not documented and the application development continued similarly. However, for Studio Manager the in-code commenting is sufficient, since the project does not have the complicated class structure. Moreover, the class and method names are self-explanatory for possible future developers of the project.

5 Discussion

This chapter explains the overall view of the application, describing the interactions between Studio Manager, the users and the local server. The ways and possibilities to set up the fully functional application from scratch are outlined. Additionally, the strengths and limitations of the whole project are described in detail.

5.1 Overall View of Studio Manager

As explained in the Introduction section, the scope of the project was limited to the company where it was undertaken. The main reason for this limitation was the rigid bond between the application and the local server. Therefore it is necessary to show the entire perspective of the application with employees as users on one side and the local server on the other. Figure 11 gives a visual representation of these connections.

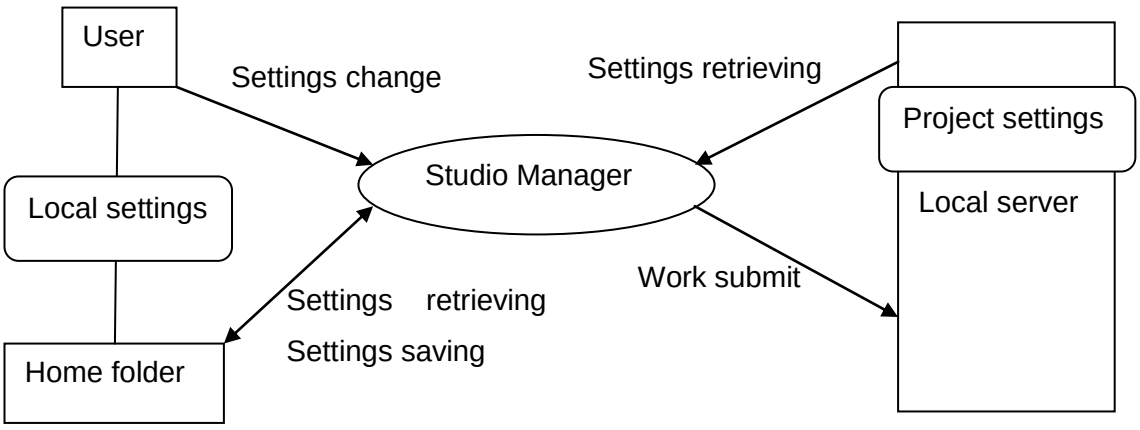

Figure 11. User-Application-Server interaction diagram.

Obviously, all the important work files and folders are located on the server. The same applies to the project settings, which become loaded at the Studio Manager launch. The user changes only the local settings, which are written to the special files in the corresponding home folder. These settings include directory remembering, sorting remembering, saving and restoring the application's window position and geometry and other similar settings. Figure 11 clearly shows how the application encapsulates the important aspects of server usage, preventing the users from direct access. Although this encapsulation system is primitive, the general idea still applies and leaves room for further development to achieve the minimal possible direct interaction of the users and the local server.

In order to outline the broad view of the application it is also important to describe the theoretical restoration of the whole system from scratch in case the local server crashes. It is vital to know exactly how Studio Manager addresses the server. Basically, all interactions between the application and the server converge to a properly named directory tree. Setting up a new local server to be fully compatible with the application would require one of the following two options to be chosen. The first way is to name the local server project tree exactly the same way as it had been set before. The second way implies tweaking the corresponding server addresses within the application. Any of these alternatives are realistic to complete within an hour. Therefore, the application-server interaction is easily restorable.

Naturally, the local machines need to be set up properly as well. As mentioned earlier in Chapter 2, Studio Manager actively uses the command line tools. These tools include additional software and libraries required for effective application operation. Although most of them are included in the default system “out of the box”, some may require manual installation. However, once a single system is configured and the application is fully functional within this system, it can be cloned to other hard drives using various cloning tools. At Algous Studio the system cloning has been successfully performed to the new machines using the CloneZilla software. Of course, the system gets duplicated and some actions still need to be undertaken to set up the newly created system. These actions include the new user creation, changing the hostname of the machine and verifying the hardware operation. Nevertheless, such actions take considerably less time than installing an entirely new operating system from scratch.

Summarizing the facts presented above, it is possible to state that the application and the environment required for its operation are fully restorable from scratch, provided the corresponding hardware is ready for installation. The amount of time needed to complete such restoration mostly depends on the number of local machines. While the project is limited to a single company, theoretically speaking, the application is transferrable to other companies. From the technical side, the possibility to transfer the application gives a variety of opportunities not only to develop the existing modules of Studio Manager, but also to tweak them in order to fit the trivia of other environments and tasks, including both project management and automation. The idea applies to all environments running Linux as the primary system and tasks related to computer graphics production.

5.2 Benefits and Possible Improvements of Studio Manager

Since the project is on-going, it is challenging to objectively evaluate its results. However, the goals that had been set before the project were successfully met. Chapter 4 gave a thorough view of the application enhancements and improvements implemented during the project. Generally, Studio Manager effectively accomplishes the assigned automation tasks and helps keeping track of studio projects by creating dailies appropriately and encapsulating the local server from direct user access.

Another important benefit of the application is its fast operation. Time is a vital resource that should always be considered and one of the application functions is to preserve as much time as possible. Also, the application is reliable to a high extent, since it is being tested by the employees on a regular basis and all possible bugs become eliminated rapidly. An essential advantage of Studio Manager is the possibility to add various modules and functionality simply by adding more tabs to the graphical user interface. In case the functionality is required, a new tab can be easily created to place all the needed widgets.

On the other hand, the application is rather demanding to preinstalled software and libraries. It simply would not start if a certain essential package was missing from the system. Another possible scenario is the lack of certain functionality, when special command line utilities are invoked by pressing a certain button. Externally it would seem that no procedure was invoked by the button, but internally an error occurs. Thus, the application could be improved by adding more GUI errors that would inform the user of possible software missing from his or her machine. In that case launching the application from the command line for further debugging could be avoided or at least reduced.

An alternative to extending the GUI error system would be creating an ultimate collection of packages needed by Studio Manager to operate properly with further automatic installation of these packages. This method could unify the application installation process and make its integration easier and more reliable. Obviously, the described method is more demanding. Fortunately, the Linux operating system is quite flexible with software installation and scripts could be used to implement this alternative method.

Generally, the error handling system is organized in a way to give a user feedback through standard output, which is the reason for a terminal to be launched together with the application. Since most errors occur if a needed package or library is missing, it is easy to trace the issue by reading the standard output messages. Most of other possible errors and warnings are displayed as GUI messages. For example, if a user presses the *Create Daily* button without selecting a sequence, a message saying *"Select the sequence!"* appears. Also, there are a few try/except blocks in the code to take care of possible runtime errors. More of these could be added in the appropriate places in further application development.

However, errors occurring within external software invoked by Studio Manager are not handled. The reason for that is encapsulation of the user interface in such ways that the user simply cannot pass inappropriate parameters to the external software. If it still happens, the problem origin lies within the application and should be fixed. Other errors related to external software can be handled by analysing the standard output messages. This error handling system could be added in future development, if its implementation was strictly required.

Although a number of graphical user interface issues were resolved during the project, the GUI could still be improved or even completely reworked. Initially the project implied simple layouts and a few buttons and combo boxes, so there was no need to design the GUI thoroughly. As the application grew larger, the GUI issues emerged on some of the tabs, which resulted in this need of implementing the new GUI. It is not an urgent matter now, but as the Studio Manager functionality increases, the issue will gain more concern. The Qt Designer software could be used to rework the existing graphical user interface to a far more appealing, consistent, and stable one.

Finally, the code organization could be improved by separating the application into several source code files with proper addressing to each other. Naturally, the number of lines of code will increase as the application gains new functionality. Therefore, keeping the code clean and structured will become more important, even though there are numerous methods with a scripting origin within the application code. Additionally, the application lacks proper documentation. The documenting issues were covered in section 4.4. However, the appropriate documentation could still be added to keep track of the code changes and to make further application development easier for programmers.

6 Conclusions

The goal of this project was to enhance and develop the project management application serving multiple purposes, mainly automation of the routine tasks within a studio making visual digital effects for movies. The application was also meant to combine various software tools to achieve the tasks. The project outcome was successful, since the application was developed and enhanced according to the initial plan. The significance of the project results can be measured by the implemented functionality and by keeping in mind the fact that the employees of the studio utilize most of the application features regularly.

The most used feature implemented during the project is the new daily script, allowing a rapid sequence to video conversion with a possibility to change multiple parameters of the output. Other implemented features include a significantly enhanced file browser with various remembering features, filtering and sequence renaming modules. Additionally, important features of powering on the render farm and controlling the permissions on the server were implemented. The graphical user interface of the application was altered as well in order to make it more appealing and comprehensible by users.

The development of the application is on-going, hence the results can be considered as intermediate, meaning more modules can be enhanced and more functionality can be added to the application. The limitations of the project include its target operating system, which is Linux only and the need of the special directory tree organization on the local server. Otherwise, the project was successful, since it was carried out within a single company for the definite purposes of this company and these purposes were reached. Apart from the new modules, future development of the project might focus on further graphical user interface enhancement and the unified installation package creation for the application.

References

1 Quiring C, Foster T. Mastering Resource Management Using Microsoft Project and Project Server 2010. Ft. Lauderdale, FL, USA: J. Ross Publishing Inc.; 2011.

2 Cobb CG. Making Sense of Agile Project Management: Balancing Control and Agility. Hoboken, NJ, USA: Wiley; 2011.

3 Shotts WE. Linux Command Line: A Complete Introduction. San Francisco, CA, USA: No Starch Press; 2012.

4 Closa D, Gardiner A, Giemsa F, Machek J. Patent Law for Computer Scientists. Berlin, Germany: Springer Berlin Heidelberg; 2010.

5 Von Hagen, W. Ubuntu Linux: Featuring Ubuntu 10.04 LTS. 3rd ed. Hoboken, NJ, USA: Wiley; 2010.

6 Lutz, M. Programming Python. 4th ed. O'Reilly Media; 2010.

7 Gift, N. From scripting to object-oriented Python programming [online]. IBM developerWorks; 14 July 2008.
URL: http://www.ibm.com/developerworks/aix/library/au-scripting_to_oo/. Accessed 19 December 2012.

8 Riverbank Computing Limited. PyQt Whitepaper [online]. 2009.
URL: http://www.riverbankcomputing.com/static/Docs/PyQt4/pyqt-whitepaper-a4.pdf. Accessed 21 December 2012.

Key Code Excerpts

The new dailies script:

```
def create_daily(self):

    server="//sun/EARTH/graphics"
    project=str(self.proj_combo.currentText())
    dailies_dir="dailies"

    kmb=str(self.kmb_combo.currentText())
    projDispName=str(self.proj_combo.currentText())
    override=str(self.versionLine.text())
    version="_V"+override
    date=str(time.localtime()[0])+"."+str(time.localtime()[1]).zfill(2)+"."+str(time.localtime()[2]).zfill(2)
    extension_mov=".mov"

    notes=unicode(self.notes.toPlainText())
    if "Type the " in notes:
        notes=""

    if not self.seqLst.selectedItems():
        item= "Select the sequence!\n"
        QMessageBox.warning(None,QString(item),QString(item))

    list_=[]
    for i in self.seqLst.selectedItems():
        list_.append(str(i.text()).split()[0])
        y=str(self.seqLst.selectedItems()[0].text())

    fileRead=str(list_[0]).replace("\\","/")
    renderpath=fileRead.split("%")[0]
    print 'renderpath', renderpath
    files = filter(os.path.isfile, glob.glob(renderpath+"*"))

    if renderpath.endswith("/"):
        files=filter(lambda i: str.isdigit(i.split("/")[-1][0]), files)

    files.sort(key=lambda x: os.path.getmtime(x))
    one_of_the_files=os.path.join(renderpath,files[-1])
    print str(one_of_the_files)

    ########### find the image size ##################
    fout = os.popen('identify ' + str(one_of_the_files))
    imageInfo = fout.read()
    size = imageInfo.split(' ')[2]
    width = int(size.split('x')[0])
    height = int(size.split('x')[1])
    ratio = float(size.split('x')[0])/float(size.split('x')[1])

    ### find the digits ######
    digits=[]
    p=re.compile('\d+')
    for everyJpeg in files:
```

```
            digits.append(int(p.findall(everyJpeg)[-1]))
        sortedDigits = sorted(digits)
        last_frame = max(sortedDigits)
        first_frame = min(sortedDigits)

        dailies_path=os.path.join(server,project,dailies_dir,date)
        daily_name=kmb+version+extension_mov
        daily=os.path.join(dailies_path,daily_name)

        first_frame=str(first_frame)
        last_frame=str(last_frame)

        if not os.path.exists(dailies_path):
            os.mkdir(dailies_path)
        else:pass

        first_frame=y.rsplit(" ",2)[1]
        last_frame=y.rsplit(" ",2)[2]
        y=y.rsplit(" ",2)[0]
        padding=int(y.split("%0")[1][0])

        fn=y.split("%0")[0]+"{"+first_frame.zfill(padding)+".."+last_frame.zfill(padding)
+"}"+y.split("%0")[1][2:]

        if self.slate.isChecked():
            slate = 1
        else:
            slate = 0

        cache = float(unicode(self.cacheComboBox.currentText()))

        fontsize=str(self.fontSpinBox.text())
        fps=str(self.fps.currentText())

        scale=str(self.resolutionComboBox.currentText())
        if scale and not scale == 'As input':
            if "/" in scale:
                width = int(int(width)*eval(scale+".0"))
                height = int(width/ratio)
            else:
                width = int(scale.split("x")[0])
                #height=int(width/ratio)          ##here either aspect height is calculated
                height = int(scale.split("x")[1])   ##or the given height from the options is forced (both should
work for normal sequences)

        cacheHeight = '0'
        if width/cache < height:
            cacheHeight = str(int((height - width/cache)/2))
        print 'cacheHeight', cacheHeight

        print 'sequence fn (with {})', fn
        print 'first frame', first_frame
        print 'padding', padding

        slateFrame = str(int(first_frame)-1)
```

```
        slatePath = renderpath + slateFrame.zfill(padding) + y.split("%0")[1][2:]
        sequence = renderpath + '%0' + str(padding) + 'd' + y.split("%0")[1][2:]

        version=self.get_version()
        project=projDispName
        date=datetime.date.today().strftime("%d-%m-%Y")
        length=str(int(last_frame)-int(first_frame)+1)

        fontToUse = '/usr/share/fonts/truetype/freefont/FreeSans.ttf'
        command="#!/bin/bash\n"
        com-
mand+='fps='+fps+';cacheHeight='+cacheHeight+';version='+version+';kmb='+kmb+';slate='+str(slate)+';l
ength='+length+';project='+project+';studio="Algous Stu-
dio";width='+str(width)+';date='+date+';a='+str(first_frame)+';height='+str(height)+';fontsize='+fontsize+
';\n'

## code omitted here – reading various parameters and adding them into the script

        #getting back the original height value before ffmpeg conversion
        command += 'let height=height+1\n\n'
        #ffmpeg starts here
        command += 'ffmpeg -y -f image2 -start_number_range 1000 -r '+fps+' -i '+sequence+' -vf "mov-
ie='+os.path.expanduser('~/canvas.png')+' [watermark]; [in]scale=$width:$height [scale];
[scale][watermark] overlay=0:0, drawtext=text=%{n}: expansion=normal: fontfile='+fontToUse+': x=w-
tw-10: y=h-lh-10: fontcolor=white: fontsize=$fontsize: shadowx=1: shadowy=1 [out]" -f mov -q:v 0 -
vcodec mjpeg '+daily+'\n'

        print command

## code omitted here – generating and executing the script

        os.system('djv_view ' + daily)        ##open the newly created daily
        if slate:
            os.remove(slatePath)        ##delete the slate frame
        if self.remember_format.isChecked():
            self.d_history("w")


The file browser enhancement methods:


class RenameDialog(QtGui.QDialog):            ##a dialog for renaming sequences
    def __init__(self,data, parent=None):

## code omitted here - defining the layout and buttons

    def print_stout(self):
## code omitted here – defining the table items
                        ##checking the user input
        if not isEnteredStringPositiveInteger(it1) or not isEnteredStringPositiveInteger(it2):
            QMessageBox.warning(None,QString("Warning!"),QString("Padding and first frame must be
positive numbers!"))
            return -1
        elif int(it1) > 9:
            QMessageBox.warning(None,QString("Warning!"),QString("Padding must be 1-9!"))
```

```
            return -1
        else:
            result=it0+' '+it1+' '+it2+' '+it3
            it.setText(result)
            self.table.resizeColumnsToContents()
            self.table.horizontalHeader().setStretchLastSection(True)
            return result

    def update_table(self):
        data=[]
        data.append(self.data)
        if data:
            self.table.setRowCount(len(data))

            for row in range(len(data)):

## code omitted here – filling the renaming table

        self.table.resizeColumnsToContents()
        self.table.horizontalHeader().setStretchLastSection(True)
        self.table.setSelectionBehavior(QtGui.QAbstractItemView.SelectRows)

    def renameSeq(self):                ##a function for renaming sequences - includes the renumber func-
tionality
        data = self.data
        old_first_frame=int(data.rsplit(" ",2)[1])
        old_last_frame=int(data.rsplit(" ",2)[2])
        numberOfFrames=old_last_frame-old_first_frame+1

        currentPath = unicode(data[:data.rfind('/')])
        oldPadding=int(data.split("%0")[1][0])

        result = self.print_stout()
        if result == -1:                ##if user enters wrong stuff as padding or first frame, the sequence is
not renamed
            return
        newBaseName = result.split(' ')[0]
        try:
            newPadding = int(result.split(' ')[1])
            newFirstFrame1 = newFirstFrame = int(result.split(' ')[2])        ##kinda counters
            extension = result.split(' ')[-1]
        except ValueError:
            QMessageBox.warning(None,QString("Warning!"),QString("Spaces found! Get rid of the spac-
es!"))
            return
        currentPath += '/'
        for i in range(numberOfFrames):

## code omitted here – intermediate renaming

        for i in range(numberOfFrames):                    ##need to do it through intermediate file
renaming (to avoid losing files)

            newIntermediateName = newBaseName
```

```
            newUltimateName = newIntermediateName + str(newFirstFrame1).zfill(newPadding) + exten-
sion
            newIntermediateName = newIntermediateName + str(newFirstFrame1).zfill(newPadding) +
'zzz' + extension
            #print 'newIN is ', newIntermediateName
            #print 'newUN is ', newUltimateName
            os.rename(currentPath+newIntermediateName, currentPath+newUltimateName)
            print 'Renamed to', newUltimateName
            #old_first_frame+=1
            newFirstFrame1+=1

        self.accept()

    def changeSeqs(self):                    ##a function for expanding the sequences into multiple
frames
        p=self.parameter                    ##and contracting the frames back into the sequence
        path=str(self.addressbar.text())
        if self.seqs_check.isChecked():                    ##if the 'sequences' checkbox is checked,
the nightmare function list_seqs
            print 'checked!'                    ##is called to list all files normally + contract the
frames into sequences if needed
            data=sorted(list_seqs(path), key=lambda item: item[p],reverse=self.sorting_order)
            self.update_file_list(data)

        else:
            print 'unchecked!'
            for name in list_seqs(path):                    ##unchecking the 'sequences'
checkbox leads to calling the listFiles function
                if name[1] == 'seq':                    ##(only in case there are sequences
in the current folder)
                    data=sorted(listFiles(path), key=lambda item: item[p],reverse=self.sorting_order)    ##list-
Files will list the frames as normal files
                    self.update_file_list(data)
                    break

    def filterList(self, dataFromInput):            ##a function for filtering files and folders
        #self.refresh_file_list(0)                ##according to the low textbar text input
        p=self.parameter
        path=str(self.addressbar.text())
        newList = []
        if self.seqs_check.isChecked():            ##filtering for sequences
            for name in list_seqs(path):
                if str(dataFromInput) not in name[0]:
                    continue
                newList.append(name)
        else:
            for name in listFiles(path):            ##filtering for frames and normal files and dirs
                if str(dataFromInput) not in name[0]:
                    continue
                newList.append(name)

        data=sorted(newList, key=lambda item: item[p],reverse=self.sorting_order)
        self.update_file_list(data)
        #print newList
```

```
    def fillTheStack(self):                    ##the function fills the stack combobox
        stackFile = open(self.stackFile, "r")          ##with the info from the stack history file

        #if os.path.getsize(self.stackFile) > 0:
        checkCounter = 0
        for i in stackFile.readlines():
            self.stack_history.addItem(unicode(i.split()[0],'utf-8'))       ##i.split()[0] means the item without
the newline symbol - \n
            checkCounter += 1
            if checkCounter > 10:       ##not more than 10 items must be loaded
                break

        stackFile.close()

    def addToStack(self):
        stackFile = open(self.stackFile, "a")

        if self.stack_history.findText(self.history[-1]) == -1 and os.path.exists(self.history[-1]):      ##if the
item is not there already and if the path is not bullshit
            self.stack_history.addItem(self.history[-1])          ##it is added to the stack
            stackFile.write(self.history[-1]+'\n')              ##and is written to the stack history file

        if self.stack_history.__len__() > 10:           ##here the maximum stack size is fixed
            self.stack_history.removeItem(0)
            stackFile = open(self.stackFile, "r")            ##
            lines = stackFile.readlines()                   ##
            lines = lines[1:]                          ##these lines are added to prevent the stack history file
            stackFile.close()                           ##from overflowing
            stackFile = open(self.stackFile, "w")            ##
            stackFile.writelines(lines)                    ##
            stackFile.close()                          ##

        stackFile.close()

    def openFromStack(self, n):                    ##a function for catching the signal of stack combobox -
changes the address
        self.addressbar.setText(n)
        self.history.append(str(n))


class SearchDialog(QtGui.QDialog):
    def __init__(self, parent=None):
      super(SearchDialog, self).__init__(parent)
      searchFormLayout = QtGui.QVBoxLayout()

## code omitted here - defining the layout and buttons

      self.cancelButton.rejected.connect(self.reject)
      self.connect(self.searchButton,QtCore.SIGNAL("clicked()"),self.doTheSearch)
      self.connect(self.browseButton,QtCore.SIGNAL("clicked()"),self.chooseDestination)
      self.connect(self.searchResultsTable,QtCore.SIGNAL("itemDoubleClicked (QTableWidgetItem *)"),
self.triggerItem)

      self.setLayout(searchFormLayout)
```

```
    def chooseDestination(self):
        f=QtGui.QFileDialog.getExistingDirectory(None,"Choose Directory",self.searchInLine.text())
        self.searchInLine.setText(str(f))

    def triggerItem(self, triggeredItem):        ##called when item is doubleclicked, writes the path to
searchOutcome for further opening
        #print 'you triggered', triggeredItem.text()
        pathTriggered = self.searchResultsTable.item(triggeredItem.row(),1).text()
        if triggeredItem.column() == 1:
            self.searchOutcome = pathTriggered
        else:
            self.searchOutcome = pathTriggered + '/' + triggeredItem.text()
        self.accept()

    def doTheSearch(self):
        while self.searchResultsTable.rowCount() > 0:            ##clear the search results
            self.searchResultsTable.removeRow(0)                 ##
        searchPath = str(self.searchInLine.text())
        searchCriteria = str(self.searchForLine.text())
        row = 0
        if searchCriteria:
            if os.path.isdir(searchPath):
                #print 'nice'
                for path, dirs, files in os.walk(searchPath):
                    for filename in files:
                        if searchCriteria.startswith('*') and not searchCriteria.endswith('*') and file-
name.endswith(searchCriteria[1+searchCriteria.rfind('*'):]):
                            if row == 9000:      ##preventing overflow
                                break
                            self.searchResultsTable.insertRow(row)              ##adding new row

                            item1 = QtGui.QTableWidgetItem(unicode(filename))    ##defining the items
                            item2 = QtGui.QTableWidgetItem(unicode(path))        ##

                            self.searchResultsTable.setItem(row,0,item1)        ##putting items into the table
                            self.searchResultsTable.setItem(row,1,item2)        ##

                            item1.setFlags(Qt.ItemIsSelectable | Qt.ItemIsEnabled)   ##setting flags - users cannot
edit the search results
                            item2.setFlags(Qt.ItemIsSelectable | Qt.ItemIsEnabled)   ##
                            row += 1                             ##row increment

## code omitted here – other search conditions

            else: QMessageBox.warning(None,QString("Warning!"),QString("Invalid path!"))
        else: QMessageBox.warning(None,QString("Warning!"),QString("Enter the search criteria!"))
        self.searchResultsTable.resizeColumnsToContents()
        if row > 1000 and row < 9000:
            self.searchStatusLabel.setText(str(row) + ' items found. Too many items! Specify the search!')
        elif row == 9000:
            self.searchStatusLabel.setText('Searching stopped - over9000 items')
        else:
            self.searchStatusLabel.setText(str(row) + ' items found.')
```

```
    def removeSpaces(self, data):
        for path, dirs, files in os.walk(data, topdown=False):
            for dirname in dirs:
                print os.path.join(path,dirname)
                os.renames(os.path.join(path,dirname), os.path.join(path,dirname.replace(' ','__')))
            for filename in files:
                print os.path.join(path,filename)
                os.renames(os.path.join(path,filename), os.path.join(path,filename.replace(' ','__')))


    def convertMovToSeq(self):
        if self.formatChoice.currentText() == 'png':
            command = 'cd ' + str(self.seqDestinationLine.text()) + '&& mplayer -nolirc -vo png ' +
self.movToConvert
        elif self.formatChoice.currentText() == 'jpeg':
            command = 'cd ' + str(self.seqDestinationLine.text()) + '&& mplayer -nolirc -vo jpeg ' +
self.movToConvert
        os.system(command)
        self.convertDialog.accept()
        self.totalRefresh()

    def addTab(self):

        if self.detect_browser_tab() == 0:                                  ##adding tab according to
the current browser tab address
            self.tabButtons[self.tabButtonCount] =
QtGui.QPushButton(self.left_browser_tab.addressbar.text())
        else:
            self.tabButtons[self.tabButtonCount] =
QtGui.QPushButton(self.right_browser_tab.addressbar.text())

        self.tabsButtonLayout.addWidget(self.tabButtons[self.tabButtonCount])
        for i in self.tabButtons:
            self.connect(self.tabButtons[i],QtCore.SIGNAL("clicked()"),partial(self.openTab,
self.tabButtons[i]))    ##waiting for signal to open the tab
        self.tabButtonCount += 1

    def removeTab(self):
        #print self.tabButtonCount
        if self.tabButtonCount > 0:                                 ##if there are tabs in the layout
            widget = self.tabsButtonLayout.takeAt(self.tabButtonCount+1).widget()    ##
            #print widget                                           ##
            widget.close()                                          ##the last one is removed
            self.tabButtonCount -= 1

    def openTab(self, tabButton):
        #print tabButton.text()
        if self.detect_browser_tab() == 0:
            self.left_browser_tab.addressbar.setText(tabButton.text())
        else:
            self.right_browser_tab.addressbar.setText(tabButton.text()
```